Kryptographie: digitale Signatur

Leitung: Adrian Spalka
Autor: Sven Tantau, tantau@informatik.uni-bonn.de
Institut für Informatik III
Universität Bonn

Stand: 8. Januar 2002

Dieses Dokument ist Teil eines Seminars über Kryptographie im Rahmen des Informatik–Grundstudiums. Hierbei soll anhand des Buches *Handbook of Applied Cryptography* das Thema der Kryptographie diskutiert werden. Grundlage für diese Ausarbeitung ist das Kapitel *Digital Signatures*.

Inhaltsverzeichnis

1 Einleitung

Diese Ausarbeitung befasst sich mit dem Thema „digitale Signatur“. Diese ist das elektronisches Pendant zu der handschriftlichen Unterschrift. Die digitale Signatur bindet eine Einheit an eine Nachricht. In der Realität kann durch sie zum Beispiel festgestellt werden, dass eine eMail von einem bestimmten Anwender kommt und ob diese nicht unerlaubt durch dritte verändert wurde.

Als erstes werden im zweiten Kapitel RSA, das verwandte Rabin-Verfahren und Angriffsmöglichkeiten auf diese Signatur-Methoden besprochen.
Im dritten Teil wird das Fiat-Shamir-Verfahren erläutert.
Das vierte Kapitel beschäftigt sich mit dem „digital signatur algorithm“ DSA.
Im fünften Abschnitt werden „One-Time[1]“-Signaturen vorgestellt.
Signaturen mit zusätzlichen Funktionen, wie das „blinde Unterschreiben“ werden im sechsten Kapitel besprochen.
Das siebte Kapitel beschreibt, wie digitale Signaturen in der Praxis eingesetzt werden am Beispiel des Programms „GnuPG“ und beschäftigt sich mit der Schlüsselverwaltung.
Die letzen drei Teile bestehen aus Zusammenfassung, Ausblick und Literaturverzeichnis.

Die digitale Signatur benutzt asymmetrische Verschlüsselung, auch „Public-Key“-Verschlüsselung genannt. Dabei wird ein Schlüsselpaar S und P benutzt, wobei alles, was mit S verschlüsselt wurde, nur mit P entschlüsselt werden kann (und umgekehrt).
Wir stellen uns nun zwei Personen (A und B) vor, die miteinander sicher kommunizieren wollen. Dazu erzeugt A das Schlüsselpaar S und P. Dann wird der Schlüssel P an B übermittelt. Der Schlüssel S bleibt bei A und wird geheim gehalten. Wenn nun B eine Nachricht bekommt, und sich diese mit P von A entschlüsseln lässt, dann weiß B, dass diese Nachricht nur von A kommen kann und dass diese nicht verändert wurde (weil nur A im Besitz von S ist).
Der Nachteil bei dieser Methode ist, dass bei großen Nachrichten ein erheblicher Zeitaufwand zum Verschlüsseln notwendig ist. Desweiteren hat keiner die Möglichkeit die Nachricht zu lesen, der nicht im Besitz des Schlüssels P ist. Diese Probleme umgeht man dadurch, dass nicht die gesamte Nachricht verschlüsselt wird, sondern nur eine Prüfsumme, die durch eine geeignete Reduzierungsfunktion erstellt wurde.
Diese Funktionen müssen folgende Eigenschaften aufweisen:

[1] deutsch: einmal

Man darf nicht aus der Prüfsumme auf die Nachricht schliessen können.
Die Prüfsumme muss ein eindeutiges Urbild haben.

Verwendete Ausdrücke

- eine *(digitale) Signatur* ist eine Bitfolge, die eine andere Bitfolge an eine Einheit bindet.
- ein *Signatur Algorithmus* ist eine Arbeitsanweisung, um eine digitale Signatur zu erstellen.
- ein *Signatur Verifikations Algorithmus* ist eine Arbeitsanweisung, um die Authentizät einer digitalen Signatur zu prüfen.
- ein *Algorithmus zur Schlüssel-Generierung* ist eine Arbeitsanweisung um Schlüssel für die Signatur zu erstellen.
- ein *Signatur Mechanismus oder Protokoll* ist eine Kombination aus einem Algorithmus zur Schlüssel-Generierung und Erzeugung/Verifikation einer digitalen Signatur.

2 RSA und RSA-Verwandte Verfahren

1977 wurde am MIT (Massachusetts Institute of Technology) der asymetrische RSA-Algorithmus von Rivest, Shamir und Adleman vorgestellt, der auf Ideen von den Herren Diffie und Hellman basiert. Dieser wird bis heute in der Praxis eingesetzt und war bis zum September 2000 Patent der gleichnamigen Firma. RSAs erster Kunde war die Firma Lotus, die ihr Produkt Notes mit einer Verschlüsselungsoption ausstatten wollte.
Das Schlüsselpaar besteht aus dem Produkt von zwei Primzahlen, wobei die Sicherheit darin besteht, dass es sehr einfach ist zwei Primzahlen miteinander zu multiplizieren, aber es bei einer gewissen Zahlengrösse sehr zeitaufwendig ist, dieses Produkt zu faktorisieren.

2.1 das RSA-Verfahren

2.1.1 RSA-Schlüsselerzeugung

Algorithmus:
A muss einen geheimen Schlüssel und einen öffentlichen Schlüssel erzeugen. Die Sicherheit hängt davon ab, wie gross p und q sind.

1. Wähle zwei Primzahlen p und q.
2. Berechne $n = p * q$ und $\phi = (p-1)(q-1)$.
3. Suche e mit $1 < e < \phi$, so dass e Teilerfremd zu ϕ ist.
4. Suche d mit $1 < d < \phi$, so dass gilt, dass $(e * d) \mod \phi = 1$.
5. Der geheime Schlüssel ist (n, d); der öffentliche Schlüssel ist (n, e).

2.1.2 RSA-Signatur und Verifikation

Algorithmus:
A muss eine Nachricht $m \in M$ signieren. Die Funktion R wandelt m in ein geeignetes Format um.
1. Berechne $\tilde{m} = R(m)$, eine Integer Zahl im Bereich $[0, n-1]$.
2. Berechne $s = \tilde{m}^d \mod n$.
3. s ist die Signatur für die Nachricht m.

B muss die Signatur $s \in S$ von A überprüfen und die Nachricht m wiederherstellen.
1. Berechne $\tilde{m} = s^e \mod n$.
2. Prüfe ob $\tilde{m} \in M : R$; wenn nicht verwerfe die Daten.

3. Stelle $m = R^{-1}\tilde{m}$ her.

2.1.3 RSA-Beispiel

Schüsselerzeugung:
$p = 3$ $q = 17$
$n = 3 * 17 = 51$, $\phi = (3-1) * (17-1) = 32$
$e = 3$ (3 und 32 sind relativ prim/teilerfremd).
$d = 11$ (weil: $3 * 11 = 33$, $33/32 = 1 \mod 1$).
geheimer Schlüssel: $(51, 11)$
öffentlicher Schlüssel:$(51, 3)$

Signieren:
wir wählen $m =$„abc“ wobei gilt: $a = 1$, $b = 2$, $c = 3$.
$a = 1$ $\quad 1^{11} = 1 = 0 * 51 + 1$
$b = 2$ $\quad 2^{11} = 2048 = 40 * 51 + 8$
$c = 3$ $\quad 3^{11} = 177147 = 3473 * 51 + 24$
s=$(1, 8, 24)$

Verifikation und Entschlüsselung:
$1^3 = 1 = 0 * 51 + 1$
$8^3 = 512 = 10 * 51 + 2$
$24^3 = 13824 = 271 * 51 + 3$
$m = (1, 2, 3) =$„abc“

2.1.4 RSA-Reblocking Problem

Eine Anwendung von RSA ist eine Nachricht zu signieren und dann zu verschlüsseln. Dabei muss die relative Grösse von „n“ beachtet werden, da es sein kann, dass wenn $(n * A) > (n * B)$ ist, die Nachricht nicht wiederhergestellt werden kann, weil die Signatur größer als der modulo nB ist.
Die Wahrscheinlichkeit dafür liegt bei 12%.

Es gibt verschiedene Möglichkeiten, dieses Problem zu lösen.
1. *Umordnung:* Dabei wird geprüft, welches n größer ist und danach vorgegangen. Es ist allerdings nicht unbedingt erwünscht, dass eine Nachricht erst verschlüsselt und dann signiert wird, da eine dritte Partei die Signatur durch eine eigene ersetzen könnte.
2. *zwei Schlüssel pro Einheit:* Die Idee hierbei ist, dass jeder Kommunikati-

onspartner zwei Schlüssel-Paare hat; eines zum signieren (lassen) und eines zum Verschlüsseln, wobei die das n der Signaturschlüssel immer in einem Bereich unterhalb des n vom Verschlüsselungsschlüssel ist.
3. *Vorschreiben des Formats von n:* Dabei werden p und q so gewählt, dass n eine spezielle Form hat. Dabei ist das höchstwertige Bit eine 1 und die folgenden k Bit alle 0. Die Wahl eines solchen n erfolgt per Intervallschachtelung in einem bestimmten Bereich. Dadurch wird das Problem nicht vollständig eliminiert, aber ab einem k in der Größenordnung von 100 ist die Wahrscheinlichkeit sehr gering.

2.1.5 Effizienz von RSA

Bei der Implementation des RSA-Algorithmus muss beachtet werden, dass zum Teil große Anforderungen an die Hardware gestellt werden, da das Finden von großen Primzahlen sehr Komplex ist und der Euklidische Algorithmus zur Bestimmung des $ggT()$ und das Potenzieren von mod n notwendig ist. Um diesen Algorithmus zum signieren auch auf „schwächerer“ Hardware wie eine Smartcard zu Implementieren, werden zum Beispiel die Schlüssel extern erzeugt und dann eingespielt und/oder nur eine digitale Prüfsumme (zum Beispiel ein SHA Hash) signiert.

2.2 Das Rabin-Verfahren

Das Rabin-Signatur-Verfahren ist dem RSA-Verfahren sehr ähnlich, wobei der Unterschied darin besteht, dass der Exponent e öffentlich ist. Der Einfachheit halber wählen wir $e = 2$.

2.2.1 Rabin-Schlüsselerzeugung

Algorithmus:
1. Wähle zwei Primzahlen p und q.
2. Berechne $n = p * q$.
3. Der öffentliche Schlüssel ist n, der geheime Schlüssel ist (p, q).

2.2.2 Rabin-Signatur und Verifikation

Algorithmus:
Signatur Generation:
1. $\tilde{m} = R(m)$
2. Berechne die Quadratwurzel s aus $\tilde{m}$ mod n.

3. s ist die Signatur für m.

Verifikation:
1. Berechne $\tilde{m} = s^2 \mod n$.
2. Prüfe ob $\tilde{m} \in M_R$; wenn nicht verwerfe die Daten.
3. Berechne $m = R^{-1}(\tilde{m})$.

2.2.3 Rabin Beispiel

Schlüsselerzeugung:
$p = 7$, $q = 11$
$n = 77$
Der öffentliche Schlüssel ist $n = 77$, der private Schlüssel ist $(7, 11)$.

Die Menge der Nachrichten ist$Q = (1, 4, 9, 15, 16, 23, 25, 36, 37, 53, 58, 60, 64, 67, 71)$ und $R(m) = m$.

Signieren:
$m = 23$
$\tilde{m} = m = R(m)$
Finde Quadratwurzel von $\tilde{m} \mod 77$
Wenn s eine solche Quadratwurzel Bezeichnet, dann gilt $s \equiv \pm 3(\mod 7)$ und $s \equiv \pm 1(\mod 11)$ Angenommen für $s = 10, 32, 45, 67$.
Wähle $s = 45$ (kann auch eine andere der Quadratwurzeln sein).

Verifizieren:
Berechne $\tilde{m} = s^2 \mod 77 = 23$. Weil $\tilde{m} \in \mathbb{M}_R$, wird die Signatur akzeptiert und die Nachricht $m = R^{-1}(\tilde{m}) = 23$ berechnet.

2.3 Angriffsmöglichkeiten

Jedes System zur Signatur bzw. Verschlüsselung hat seine Vor- und Nachteile. Die im folgenden beschriebenen Angriffsmöglichkeiten beziehen sich daher nicht nur auf RSA- und RSA-Verwandte Verfahren, sondern sind meistens ein generelles Problem der Kryptografie.

2.3.1 Das Faktorisierungsproblem

Die Sicherheit von RSA beruht auf dem Faktorisierungsproblem.
Dieses kann man durch zwei verschiedene Ansätze versuchen zu lösen. Entweder, man findet einen Algorithmus, der dieses Problem in einer Zeit löst, die zufriedenstellend ist, oder man benutzt viel Rechenleistung. Dadurch, dass in der heutigen Zeit und in der Zukunft immer mehr Rechner vernetzt sind, ist die Möglichkeit entstanden, viele Maschinen parallel[2] an einem Problem arbeiten zu lassen. Eine immer grössere Viren und Würmer Plage im Internet zeigt, dass die Eigentümer der Rechenanlagen nicht immer über diese verfügen. Für einige simple Internet Würmer war es möglich, Infektionsraten von 100.000 Computern pro Minute zu erreichen, was zeigt, dass man dieses Problem nicht unterschätzen sollte.
Gegen das Finden eines Algorithmus, der das Faktorisierungsproblem schnell lößt, gibt es keine Abhilfe. Man verlässt sich hiebei auf die Tatsache, dass es sehr vielen Menschen auf der ganzen Welt möglich ist, den Algorithmus einzusehen und glaubt, dass die Entdeckung entweder garnicht oder von vielen Leuten gemacht wird, was eine Geheimhaltung so gut wie unmöglich macht. Ob man in diese Sicherheit Vertrauen setzen möchte, bleibt einem selbst überlassen.
Gegen das Brechen von RSA mit Hilfe von viel Rechenleistung kann man etwas tun. Man sollte p und q in etwa gleich groß halten, $(p-1)$, $(q-1)$ sollten grosse Primfaktoren haben und $ggT(p-1, q-1)$ sollte klein sein.

2.3.2 Implementaion

Ein weiteres Problem in der Kryptografie ist die Implementation der verschiedenen Verfahren.
Im Fall von RSA wäre es zum Beispiel eine Katastrophe, wenn der geheime Schlüssel bekannt würde. Deshalb müssen alle Menschen oder Programme, die auf den Schlüssel zugreifen sollen, vertrauensvoll sein oder man muss sie von Zugriffen ausschliessen. Beim praktischen Einsatz von digitalen Signaturen geschieht das zum Beispiel mit zusätzlicher Verschlüsselung des geheimen Schlüssels.

[2] das Seti@Home Projekt ist ein Beispiel für verteiltes Rechnen

3 Das Fiat-Shamir-Verfahren

Jedes Identifikationsverfahren, das auf einer „witness-challenge-response“[3] -Sequenz aufsetzt, kann in ein Signatur-Verfahren umgewandelt werden, indem man die zufälligen Anfragen durch eine One-Way-Hash-Funktion ersetzt. Das Fiat-Shamir-Signatur-Verfahren, ist im Vergleich zu RSA wesenlich schneller[4] , wodurch der Einsatz gerechtfertigt wird.

3.1 Feige-Fiat-Shamir-Signatur-Verfahren

Dieses Verfahren benötigt eine One-Way-Hash-Funktion h, die einen Bitstrom beliebiger Grösse auf einen Bitstrom der Grösse k abbildet.

3.1.1 Schlüsselerzeugung

Algorithmus:
1. $n = p * q$, wobei p,q geheime, zufällige Primzahlen sind.
2. Wähle ein zufälliges k und verschieden zufällige Zahlen $s_1, s_2, ..., s_k \in \mathbb{Z}^*$.
3. $v_j = s_j^{-2} \mod n$, $1 \leq j \leq k$
4. Der öffentliche Schlüssel von A ist der k-Tupel $(v_1, v_2, ..., v_k)$ und n; der geheime Schlüssel von A ist der k-Tupel$(s_1, s_2, ..., s_k)$.

3.1.2 Signatur und Verifikation

Signatur Generation:
Algorithmus:
1. Wähle eine Zufallszahl r, $1 \leq r \leq n-1$.
2. $u = r^2 \mod n$
3. $e = (e_1, e_2, ..., e_k) = h(m||u)$; mit $e_i \in \{0, 1\}$
4. $s = r * \prod_{j=1}^{k} s_j^{e_j} \mod n$
5. Die Signatur für die Nachricht m ist (e, s).

Verifikation:
Algorithmus:
1. $w = s^2 * \prod_{j=1}^{k} * v_j^{e_j} \mod n$
2. $e' = h(m||w)$
3. Wenn $e = e'$ ist die Signatur zu akzeptieren.

[3] A stellt B eine Reihe von Aufgaben, die zu lösen sind. Zusätzlich wird eine dritte Partei zur Kontrolle benötigt.

[4] benötigt nur $1 - 4\%$ der modularen Operationen

3.2 Zusätzliches

Im Gegensatz zu RSA-Signatur können alle Parteien den gleichen Modulus n benutzen. Um die Sicherheit zu bewahren, muss eine dritte vertrauenswürdige Partei (TTP[5]) die notwendigen Schlüssel für die Kommunikationspartner erstellen.

[5]TTP steht für trusted third „party“

4 DSA und DSS

DSA steht für Digital-Signature-Algorithm und ist eine Variante des El-Gama-Systems. Er wurde 1991 vom amerikanischen National Institute of Standards and Technology (NIST) entwickelt und ist seit 1994 dort Standard (FIPS 186), auch DSS (Digital Signature Standard) genannt.

4.1 DSA

Es wird eine Hash-Funktion $h : \{0,1\}- > \mathbb{Z}_q$ benötigt, wobei der DSS die Verwendung des Secure- Hash-Algorithm 1 (SHA-1) vorschreibt, der eine 160 Bit-Zeichenkette produziert.

4.1.1 Schlüsselerzeugung

Algorithmus:
1. Wähle Primzahl q, wobei $2^{159} < q < 2^{160}$.
2. Wähle t mit $0 \leq t \leq 8$ und wähle Primzahl p mit $2^{511+64t} \leq p \leq 2^{512+64t}$ mit der Eigenschaft, dass q $(p-1)$ teilt.
3. Wiederhole bis $\alpha \neq 1$: suche $g \in \mathbb{Z}_p^*$ und berechne $\alpha = g^{(p-1)/q} \mod p$.
4. Suche Zufallszahl a mit $1 \leq a \leq q-1$.
5. $y = a^{\alpha} \mod p$
6. Der öffentliche Schlüssel ist (p, q, α, y), der geheime a.

4.1.2 Signatur und Verifikation

Algorithmus:
A signiert m:
1. Wähle eine geheime Zufallszahl k, mit $0 < k < q$.
2. $r = (\alpha^k \mod p) \mod q$
3. Berechne $k^{-1}(\mod q)$.
4. $s = k^{-1}\{h(m) + ar\} \mod q$
5. Die Signatur für m ist (r, s).

B verifiziert die Signatur (r, s) von m:
1. Prüfe, ob $0 < r < q \wedge 0 < s < q$: wenn nicht, ist die Signatur zu verwerfen.
2. $w = s^{-1} \mod q$
3. $u_1 = w * h(m) \mod q$; $u_2 = rw \mod q$
4. $v = (\alpha^{u_1} y^{u_2} \mod p) \mod q$

5. Nur wenn $v = r$ ist die Signatur zu aktzeptiren.

4.1.3 Beispiel

Ein Beispiel mit kleinen Zahlen:
Schlüssel erzeugen:
$q = 3 \wedge p = 7$ mit q teilt $(p - 1)$
für $g = 3$ gilt $\alpha = 2$
$a = 2$
$y = \alpha^a = 4$
Der öffentliche Schlüssel ist (p, q, α, y), der Private ist a.

Signatur erzeugen:
Um es einfach zu machen: $m = 12 \wedge h(m) = 12$
$k = 2$
$r = \alpha^k \mod q = 1$
$k^{-1}(\mod q) = 4$
$k^{-1} \mod 3 = 1$
$s = 1 * (12 + 2 * 1) \mod 3 = 2$
Die Signatur für m ist (r, s)

Verifizieren:
$w = 4 \mod 3 = 1$
$u_1 = 0$
$u_2 = 1$
$v = 1 = r$
da $v = r$ wird die Signatur akzeptiert.

4.2 DSS

Der DSS (Digital Signature Standard) benutzt den DSA und schreibt genau vor, welche Algorithmen zum Beispiel für das Finden von Zufallszahlen zu verwenden sind. Zudem wird eine gewisse Schlüssellänge vorgeschrieben, beziehungweise die Primzahl p darf die Grösse von 1024 Bit[6] nicht überschreiten.

[6] [FIPS 186] liefert Details

4.3 Die Sicherheit von DSA und DSS

Die Sicherheit des DSA unterliegt dem Problem des diskteten Logarithmus. Wenn man davon ausgeht, dass es keine schnellere Möglichkeit für einen Angreifer gibt, s im Bezug auf m zu fälschen, dann muss der Angreifer alle s ausprobieren. Die Wahrscheinlichkeit, dass dieses gelingt liegt bei $(1/p)$. Die Tatsache, dass DSA und DSS ein p größer als 1024 Bit verbieten, sollte daher sehr kritisch betrachtet werden.

Desweiteren sollte darauf geachtet werden, dass $(p-1)$ grosse Primfaktoren hat, wodurch ein Angriff auf den diskreten Logarithmus durch die Pohlig-Hellman-Methode erschwert wird.
Es muss für jede Nachricht m ein neues zufälliges k berechnet werden, da es sonst durch Kryptoanalyse einfach wäre, den geheimen Schlüssel zu ermitteln.

Die Hash Funktion, ist sehr relevant für die Sicherheit des DSA. Falls keine Funktion zum hashen der Nachricht benutzt würde, ist es einem Angreifer möglich ein beliebiges Paar (u, v) zu wählen mit $ged(v, p-1) = 1$. Er berechnet: $r = a^u * y^v \mod p = \alpha^{u+ay} \mod p$ und $s = -rv^{-1} \mod (p-a)$. (r, s) ist eine gültige Signatur für die Nachrich $m = su \mod (p-1)$, da $(\alpha^m \alpha^{-ar})^{s^{-1}} = \alpha^u y^v = r$.

4.4 DSA Geschwindigkeit

Im Vergleich zu RSA, ist das Generieren von Schlüsseln etwa gleich schnell. Die Verifizierung der Signatur erledigt RSA 10 bis 40 mal schneller als DSA. Einzig bei der Schlüsselerzeugung ist DSA schneller, was in der Praxis jedoch nicht so oft benötigt wird.
Es muss zu diesen Daten angemerkt werden, dass es sehr stark auf die Parameter ankommt, unter denen ein Geschwindigkeitstest gemacht wird, so dass die Aussagekraft solcher Test hinterfragt werden sollte. In der praktischen Anwendung sind die beiden Algorithmen etwa gleichschnell.

5 One-Time-Signaturen

Eine One-Time[7]-Signatur-Funktion erlaubt das Signieren genau einer Nachrich; sonst könnte die Signatur gefälscht werden. Da für jede zu signierende Nachricht ein neuer öffentlicher Schlüssel erstellt werden muss, ist diese Form der Signatur in der Praxis selten üblich. Der in diesem Kapitel behandelte Algorithmus von Merkle umgeht dieses Problem dadurch, dass die Schlüsselerzeugung an eine dritte, vertrauenswürdige Partei übertragen wird. Ein Vorteil von One-Time-Signaturen ist, dass diese im Vergleich zu anderen Methoden schnell sind.

5.1 Das Rabin-Verfahren

Das One-Time-Signatur-Verfahren von Rabin war eines der ersten Ansätze für die Realisierug der digitalen Signatur. Bei der Verifikation wird erst die Authentizität des Absenders überprüft und dann die Authentizität der Nachricht. Da nur das Signieren einer einzigen Nachricht erlaubt ist und das Verifizieren eine Interaktion zwischen A und B bedingt, ist diese Methode in der Praxis nicht üblich.

5.1.1 Schlüsselerzeugung

Algorithmus:
1. Wähle ein symmetrisches Verschlüsselungsverfahren E (z.B. DES[8]).
2. K sei eine Menge von l-Bit Zeichenketten. Erzeuge $2n$ zufällige geheime Zeichenketten $z_1, z_2, ..., z_{2n} \in K$, mit der jeweiligen Bitlänge l.
3. $y_i = E_{k_i}(M_0(i)), 1 \leq i \leq 2n$, wobei M_0 alle 0-Zeichenfolgen der länge l angibt.
4. Der öffentliche Schlüssel ist $(y_1, y_2, ..., y_{2n})$, der private Schlüssel ist $(k_1, k_2, ..., k_{2n})$.

5.1.2 Signatur und Verifikation

Signatur:
Algorithmus:
1. $y_i = E_{k_i}(h(m)), 1 \leq i \leq 2n$
2. Die Signatur für m ist $(s_1, s_2, ..., s_{2n})$.

Verifikation:

[7] deutsch: „einmal“
[8] Data Encryption Standard

Algorithmus:
1. Wähle n zufällige Zahlen r_j, $1 \leq r_j \leq 2n$, mit $1 \leq j \leq n$.
2. Fordere beim Absender die Schlüssel k_{r_j} an, $1 \leq j \leq n$.
3. Überprüfe die Authentizität der Schlüssel durch Berechnen von: $z_j = E_{k_{r_j}}(M_0(r_j))$ und dem Test ob $z_j = y_{r_j}$ für jedes $1 \leq j \leq n$.
4. Überprüfe ob $s_{r_j} = E_{k_{r_j}}(h(m))$, $1 \leq j \leq n$.

5.2 das Merkle Verfahren

Der One-Time-Algorithmus von Merkle unterscheidet sich vom Rabin-Algorithmus in der Hinsicht, dass die notwendige Interaktion zwischen den kommunizierenden Parteien wegfällt und das Verifizieren von einer dritten Partei übernommen wird.

5.2.1 Schlüsselerzeugung

Algorithmus:
1. Wähle $t = n + \lfloor \lg n \rfloor + 1$ zufällige geheime Zeichenketten $k_1, k_2, ..., k_t$, mit der jeweiligen Bitlänge l.
2. $v_i = h(k_i), 1 \leq i \leq t$
3. Der öffentliche Schlüssel ist $(v_1, v_2, ...v_t)$, der private $(k_1, k_2, ...k_t)$.

5.2.2 Signatur und Verifikation

Signatur:
Algorithmus:
1. Berechne c, die binäre Representation für die Anzahl der "0en" in m.
2. Bilde $w = m||e = (a_1, a_2, ..., a_t)$.
3. Bestimme die Position $i_1, i_2, ..., i_u$ in w, so dass $a_{i_j} = 1$, $1 \leq j \leq u$.
4. Sei $s_j = k_{i_j}$, $1 \leq j \leq u$.
5. Die Signatur für die Nachricht m ist $(s_1, s_2, ..., s_u)$.

Verifikation:
Algorithmus:
1. Berechne c, die binäre Representation für die Anzahl der "0en" in m.
2. Bilde $w = m||c = (a_1, a_2, ..., a_t)$.
3. Bestimme die Position $i_1, i_2, ..., i_u$ in w, so dass $a_{i_j} = 1$, $1 \leq j \leq u$.
4. Die Signatur ist zu Akzeptieren, wenn $v_{i_j} = h(s_j)$ für alle $1 \leq j \leq u$.

6 Signaturen mit zusätzlichen Funktionen

Signaturen, wie sie bis jetzt vorgestellt wurden, finden in der Praxis viele Anwendungsgebiete. Dennoch bedarf es zusätzlicher Protokolle und Methoden, um Probleme wie zum Beispiel Vertragssicherheit oder Datenschutzgesetze in den Griff zu bekommen.

Meistens werden herkömmliche Methoden zur digitalen Signatur mit den verschiedenen Protokollen kombiniert, die notwendig sind.

6.1 Blindes Unterschreiben

In der Praxis kann es vorkommen, dass A möchte, dass B eine Nachricht m für ihn signiert, ohne dass B den Inhalt der Nachricht kennt.
Dies kann zum Beispiel der Fall sein, wenn ein Angestellter möchte, dass seine Firma seinen Arbeitsplatz bestätigt ohne dass diese seine Kommunikation mit Behörden nachvollziehen kann.

Es muss erwähnt werden, dass es nicht ausreicht, den Hash-Wert $h(m)$ der Nachricht an B zu schicken und unterschreiben zu lassen, da es theoretisch möglich ist, die Nachricht durch Suchen und Hashen zu identifizieren.

Dieses Problem wird dadurch gelöst, dass die Nachricht m durch eine Zufallszahl k verschleiert wird. Diesen Vorgang nennt man „blinding“.

6.1.1 Blinde Signatur nach Chaum

Chaum benutzt in seinem Protokoll RSA. $f()$ und $g()$ sind die blinding- und unblinding-Funktionen. $S_B(x)$ stellt die Signatur von x durch B dar. Es muss gelten: $g(S_B(f(m))) = S_B(m)$ wobei $f()$ und $g()$ nur dem Sender A bekannt sein dürfen. Der öffentliche Schlüssel von B ist (n, e) der private (n, d).

Wenn A eine “blinde Signatur“ von B möchte, geht er folgendermassen vor:
1. A berechnet $m^* = mk^e \mod n$ wobei $0 \leq k \leq n-1$ und schickt m^* an B.
2. Blinding: B berechnet $s^* = (m^*)^d \mod n$ und schickt s^* an A.
3. Signieren: A berechnet $s = k^{-1}s^* \mod n$.
4. Unblinding: s ist die Signatur von m durch B.

6.2 Verbindliches Unterschreiben

Verbindliche[9] Signaturen geben die Möglichkeit, dass A nur dann beweisen kann, dass eine Signatur s zu einer Nachricht m von B kommt, wenn B bei der Verifikation mitwirkt.
Die Möglichkeit besteht, dass B sich weigert die Signatur zu bestätigen, bewusst Fehler bei der Verifikation macht oder die Signatur gefälscht ist. Deshalb wurde das "Disavowal[10]"-Protokoll entwickelt, mit dem man die letzen beiden Möglichkeiten beweisen kann. Wenn B einen Verifikations-Prozess ablehnt, wird eine gültige Signatur angenommen.

6.2.1 Schlüsselerzeugung

Algorithmus:
1. Wähle $p = 2q + 1$, wobei p,q Primzahlen sind.
2. Wähle zufällig $beta \in \mathbb{Z}_p^*$ und berechne $\alpha = \beta^{(p-1)/q} \mod p$ bis $\alpha \neq 1$.
3. Wähle zufällig $a \in \{1, 2, ..., q-1\}$ und berechne $y = \alpha^a \mod p$.
4. Der öffentliche Schlüssel ist (p, α, y), der private ist a.

6.2.2 Signatur und Verifikation

Signieren:
Algorithmus:
1. $s = m^a \mod p$
2. s ist die Signatur zu m.

Verifizieren:
Algorithmus und Protokoll:
A möchte die Signatur von B überprüfen.
1. A wählt zufällig geheime $x_1, x_2 \in \{1, 2, ..., q-1\}$.
2. A berechnet $z = s^{x_1} y^{x_2} \mod p$ und sendet z zu B.
3. B berechnet $w = (z)^{a^{-1}} \mod p$ und sendet w zu A.
4. A berechnet $w' = m^{x_1} \alpha^{x_2} \mod p$.
5. Wenn $w' = w$, akzeptiert A die Signatur.

[9] oft auch Undeniable Signatures genannt
[10] übersetzbar mit: Ableugnen

6.2.3 „Disavowal“-Protokoll

Mit diesem Protokoll kann A feststellen, ob die Signatur s zur Nachricht m von B kommt, ob B versucht die Signatur zu leugnen oder ob die Signatur gefälscht wurde.

Algorithmus und Protokoll:
1. A wählt zufällig geheime $x_1, x_2 \in \{1, 2, ..., q-1\}$.
2. A berechnet $z = s^{x_1} y^{x_2} \mod p$ und sendet z zu B.
3. B berechnet $w = (z)^{a^{-1}} \mod p$ und sendet w zu A.
4. A berechnet $w' = m^{x_1} \alpha^{x_2} \mod p$.
5. Wenn $w' = w$, akzeptiert A die Signatur und das Protokoll stoppt.
6. A wählt zufällig geheime $x_1', x_2' \in \{1, 2, ..., q-1\}$
7. A berechnet $z' = s^{x_1'} y^{x_2'} \mod p$ und sendet z' zu B.
8. B berechnet $v = (z')^{a^{-1}} \mod p$ und sendet v zu A.
9. A berechnet $v' = m^{x_1'} \alpha^{x_2'} \mod p$.
10. Wenn $v = v'$, akzeptiert A die Signatur und das Protokoll stoppt.
11. A berechnet $c = (w\alpha^{x_2})^{x_1'} \mod p$ und $c' = (v\alpha^{-x_2'})^{x_1} \mod p$. Wenn $c = c'$ ist s eine Fälschung, sonst versucht B die Signatur zu leugnen.

6.3 Fail-Stop Signaturen

Fail-Stop Signaturen von den Herren van Heyst und Pederson bieten die Möglichkeit, Fälschungen der eigenen Signatur zu beweisen oder dem Vorwurf, dass man eine bestimmte Nachricht nicht signiert hat, entgegenzutreten.

6.3.1 Schlüsselerzeugung

Die Schlüsselerzeugung geschieht durch A und einer dritten vertrauenswürdigen Partei (TTP[11]).

Algorithmus:
TTP:
1. Wähle Primzahlen p und q, so dass $(p-1)$ von q geteilt wird.
2. Wiederhole bis $\alpha \neq 1$: wähle zufällig $g \in \mathbb{Z}_p^*$ und berechne $\alpha = g^{(p-1)/q}$.
3. Wähle zufällig und geheim a, $1 \leq a \leq q-1$ und berechne $\beta = \alpha^a \mod p$.
4. Schicke (p, q, α, β) an A.

[11] Trusted Third Party

A:
5. Wähle geheime Zufallszahlen x_1, x_2, y_1, y_2 aus dem Intervall $[0, q-1]$.
6. $\beta_1 = \alpha^{x_1}\beta^{x_2}$ $\beta_2 = \alpha^{y_1}\beta^{y_2} \mod p$
7. Der öffentliche Schlüssel ist $(\beta_1, \beta_2, p, q, \alpha, \beta)$, der private $\bar{x} = (x_1, x_2, y_1, y_2)$

6.3.2 Signatur und Verifikation

Signatur:
Algorithmus:
1. $s_{1,m} = x_1 + my_1 \mod q$ und $s_{2,m} = x_2 + my_2$ wobei $m \in [0, q-1]$
2. Die Signatur der Nachricht m ist $(s_{1,m}, s_{2,m})$.

Verifikation:
Algorithmus:
1. $v_1 = \beta_1\beta_2^m \mod p$ und $v_2 = \alpha^{s_{1,m}}\beta^{s_{2,m}} \mod p$
2. Die Signatur ist valid, wenn $v_1 = v_2$.

6.3.3 Fälschungsbeweis

Hintergrund:
Wir nehmen den öffenlichen Schlüssel $(\beta_1, \beta_2, p, q, \alpha, \beta)$ und den privaten Schlüssel $\bar{x} = (x_1, x_2, y_1, y_2)$.
Es gibt genau q^2 mal ein $\bar{x}' = (x_1', x_2', y_1', y_2')$ mit $x_1', x_2', y_1', y_2' \in \mathbb{Z}_{11}$, die als Ergebnis den gleichen Anteil (β_1, β_2) des öffentlichen Schlüssels haben.
Sei T die Menge aller dieser $\bar{x}'$. Für jedes $m \in \mathbb{Z}_q$, gibt es genau q mal ein $\bar{x}'$, welches die gleiche Signatur für m liefert.
Sei $m' \in \mathbb{Z}_q$ ungleich m. Dann ergeben die q mal in T enthaltenen $\bar{x}'$, die die gleiche Signatur für m ergeben, q unterschiedliche Signaturen für m'.
Daher beträgt die Wahrscheinlichkeit, dass eine gefälschte Signatur erzeugt wird $q^2/q = 1/q$.
Wenn man nun davon ausgeht, dass ein Angreifer doch eine Signatur gefälscht hat, dann kann mit folgendem Algorithmus dieses mit hoher Wahrscheinlichkeit nachgewiesen werden. Es wird versucht, aus der vermeintlich gefälschten Signatur a zu errechnen. Wenn dies gelingt, ist das ein Beweis für eine Fälschung, da nur die dritte Partei dieses kennen darf.

Algorithmus:
1. s' sei die gefälschte Signatur für m. Solange $s = s'$: berechne s für m
2. $a = (s_{1,m} - s_{1,m}') * (s_{2,m} - s_{2,m}')^{-1} \mod q$

7 Praxis

In diesem Kapitel werden die Grundprinzipien der Schlüsselverwaltung besprochen, das Programm „GnuPG“mit seinen Funktionen zur digitalen Signatur vorgestellt und über die Probleme, die in der Praxis entstehen berichtet.

7.1 Schlüsselverwaltung

Damit sich zwei Parteien (A und B) per eMail digital signierte Daten schicken können, ist es notwendig, dass die öffentlichen Schlüssel ausgetauscht werden. Damit A sich sicher sein kann, dass der Schlüssel, den er bekommen hat auch von B ist, ist es notwendig, dass B zum Beispiel seinen Schlüssel ausdruckt und bei A persönlich abgibt. Grade bei Internet-eMails ist es unter Umständen sehr einfach, einen Schlüssel abzufangen und verändert weiterzuleiten.
Die Lösung für dieses Problem liegt im Problem.
Wenn die Schlüssel signiert sind ist es möglich zu überprüfen, ob die Daten geändert wurden.
Beispiel:
A, B, C tauschen auf einem Treffen ihre Schlüssel P_A, P_B, P_B aus und zeigen sich gegenseitig ihren Ausweis um sich über die Identität des anderen absolut sicher zu sein. Dann signieren sie gegenseitig ihre Schlüssel[12].
Später trifft A auf D und die Schlüssel werden wieder ausgetauscht.
Wenn jetzt D mit B Kommunizieren möchte, kann sich D den Schlüssel von B (der von A signiert ist) per eMail schicken lassen und mit dem Schlüssel von A auf Veränderungen prüfen. Eine perfekte Sicherheit ist dadurch noch nicht erreicht, da D nicht weiß, unter welchen Umständen A den Schlüssel von B signiert hat.
Es gibt unter Computerbenutzern öfters Treffen, die unter dem Motto stehen, Schlüssel auszutauschen und so ein Netz des Vertrauens zu schaffen.

Je mehr Leute den eigenen Schlüssel signiert haben, desto vertrauenswürdiger wird dieser. In Deutschland gibt es verschiedene Institutionen, die kostenlos oder gegen Gebühr den öffentlichen Schlüssel von Personen oder Systemen

[12]den eigenen Schlüssel zu signieren bringt für den Empfänger und Absender mehr Sicherheit im Bezug darauf, ob am Zugang des Computerbenutzers im Bezug auf die UID manipuliert wurde

signieren und mit ihrem Namen den Schlüssel vertrauenswürdiger machen.

7.2 GnuPG

„GnuPG[13]“ ist ein Kryptografie-Programm, das auf den unterschiedlichsten Computerarchitekturen und Betriebssystemen lauffähig ist. Es arbeitet nach dem „OpenPGP“-Standard RFC 2440 und es steht jedem frei, den Quellcode zu analysieren und so Fehler zu finden.

„GnuPG“ ist ein sogenanntes komandozeilenorientiertes Programm, für das es verschiedenste grafische Benutzeroberflächen gibt. Dadurch ist ein schnelles und nahezu automatisiertes Arbeiten möglich.

Um „GnuPG“ besser kennenzulernen zeige ich im folgenden, wie das Programm auf einem Linux-System installiert wird und wie man damit Dateien signiert. Bildschirmausgaben, die nicht relevant sind, sind nicht aufgeführt.

7.2.1 Installation und Anwendung

Beziehen kann man das Programm direkt von der Internet Seite der Programmierer unter: http://www.gnupg.org , desweiteren ist es oft bei Linux-Distributionen enthalten.

```
conrad:/usr/local # tar xfzv gnupg-1.0.6.tar.gz
```

Mit dem „tar“-Kommando wird die Datei „gnupg-1.0.6.tar.gz“ entpackt und in das Verzeichnis „gnupg-1.0.6“ kopiert.

```
conrad:/usr/local # cd gnupg-1.0.6
```

Das „cd“-Kommando wechselt das Verzeichnis.
„ls“ zeigt die Dateien an.

```
conrad:/usr/local/gnupg-1.0.6 # ls
.          ChangeLog    README        aclocal.4     debian   po
..         INSTALL      THANKS        checks        doc      scripts
```

[13]GNU Privacy Guard

```
ABOUT-NLS  Makefile.am  TODO          cipher        g10      stamp-h.in
AUTHORS    Makefile.in  VERSION       config.h.in   include  tools
BUGS       NEWS         acconfig.h    configure     intl     util
COPYING    PROJECTS     acinclude.m4  configure.in  mpi      zlib
conrad:/usr/local/gnupg-1.0.6 #
```

Natürlich sollte jeder die Dateien README, INSTALL und BUGS lesen.
In der Datei INSTALL sind Parameter zum Konfigurieren enthalten, die von Vorteil sein können.
Für eine Standard-Installation geht man folgendermassen vor:

```
conrad:/usr/local/gnupg-1.0.6 # ./configure
conrad:/usr/local/gnupg-1.0.6 # make
conrad:/usr/local/gnupg-1.0.6 # make install
conrad:/usr/local/gnupg-1.0.6 # cd /root
conrad:~ #
```

Durch diese drei Befehle wird erst das System bzüglich Interner Parameter befragt, dann der Quellcode kompiliert und am Schluss in die richtigen Verzeichnisse kopiert.

Es ist wichtig, dass das Programm mit den Benutzerrechten des root-Users läuft, da sonst sensible Daten auf der Swap-Partition[14] ausgelagert werden können. Bei Multi-User-Systemen wie UNIX stellt die Option des „setuid“, mit der man einem Programm root-Rechte zuweisst, eine potentielle Sicherheitslücke[15] dar.
Aus diesem Grund entfällt der Schritt bei dieser Installation.

Als nächstes muss ein Schlüsselpaar generiert werden.

```
conrad:~ # gpg --gen-key
gpg (GnuPG) 1.0.6; Copyright (C) 2001 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

Please select what kind of key you want:
```

[14] ein Bereich der Festplatte, der das RAM entlastet.

[15] wenn ein Programm mit root-Rechten von einem Anwender gestartet werden kann und dieser das Programm dazu benutzt System-Kommandos auszuführen, geschieht das mit root-Rechten.

```
   (1) DSA and ElGamal (default)
   (2) DSA (sign only)
   (4) ElGamal (sign and encrypt)
Your selection? 1
[.........Eingabe duch Benutzer.........]
DSA keypair will have 1024 bits.
About to generate a new ELG-E keypair.
              minimum keysize is  768 bits
              default keysize is 1024 bits
    highest suggested keysize is 2048 bits
What keysize do you want? (1024) 1024
[.........Eingabe duch Benutzer.........]
Requested keysize is 1024 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0) 1
.........Eingabe duch Benutzer.........]
Key expires at Wed Jan 3 05:15:26 2002 CET
Is this correct (y/n)? y
[.........Eingabe duch Benutzer.........]
You need a User-ID to identify your key; the software constructs the user id
from Real Name, Comment and Email Address in this form:
    "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Real name: Sven Tantau
Email address: tantau@informatik.uni-bonn.de
[.........Eingabe duch Benutzer.........]
Comment: testkey
    "Sven Tantau (testkey) <tantau@informatik.uni-bonn.de>"
[.........Eingabe duch Benutzer.........]
Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit?
[.........Eingabe duch Benutzer.........]
You need a Passphrase to protect your secret key.

Enter passphrase: etw4z m3rl<Bares_nischzuraten
[.........Eingabe duch Benutzer.........]
We need to generate a lot of random bytes. It is a good idea to perform
```

```
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
++++++++++...++++++++++++++++++++++++.++++++++++++++
Not enough random bytes available.  Please do some other work to give
the OS a chance to collect more entropy! (Need 284 more bytes)
public and secret key created and signed.
conrad:~ #
```

Als erstes sollte nun ein „Wiederrufs-Zertifikat“ erzeugt werden. Es gibt dem Besitzer die Möglichkeit den Schlüssel glaubwürdig für ungülgit erklärenden zu lassen, sollte dieser verlöhren gehen.

```
conrad:~ # gpg --gen-revoke tantau

sec  1024D/F5F93304 2002-01-03   Sven Tantau (testkey) <tantau@informatik.uni-bo

Create a revocation certificate for this key? yes
Please select the reason for the revocation:
  1 = Key has been compromised
  2 = Key is superseded
  3 = Key is no longer used
  0 = Cancel
(Probably you want to select 1 here)
Your decision? 3
Enter an optional description; end it with an empty line:
> nur ein test
>
Reason for revocation: Key is no longer used
nur ein test
Is this okay? yes

You need a passphrase to unlock the secret key for
user: "Sven Tantau (testkey) <tantau@informatik.uni-bonn.de>"
1024-bit DSA key, ID F5F93304, created 2002-01-03

Enter passphrase:
ASCII armored output forced.
Revocation certificate created.

Please move it to a medium which you can hide away; if Mallory gets
```

```
access to this certificate he can use it to make your key unusable.
It is smart to print this certificate and store it away, just in case
your media become unreadable.  But have some caution:  The print system of
your machine might store the data and make it available to others!
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.0.6 (GNU/Linux)
Comment: For info see http://www.gnupg.org
Comment: A revocation certificate should follow

iFUEIBECABUFAjxNfGsOHQNudXIgZWluIHRlc3QACgkQFQV2RfX5MwQ+pACfesXA
AUqUP1jDbBEkemExkb+Kx9EAoK9N/6hbWbrA2FfLN36MNkIBd+qC
=7Whn
-----END PGP PUBLIC KEY BLOCK-----
```

Im folgenden wird die Datei „eMail“ auf unterschiedlichste Weise mit einer Signatur versehen und am Ende auf Gültigkeit geprüft.

```
conrad:~ # gpg --clearsign eMail

You need a passphrase to unlock the secret key for
user: "Sven Tantau (testkey) <tantau@informatik.uni-bonn.de>"
1024-bit DSA key, ID F5F93304, created 2002-01-03

conrad:~ # cat eMail.asc
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

Ich bin eine eMail!
Bitte signier mich!

-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.0.6 (GNU/Linux)
Comment: For info see http://www.gnupg.org

iEYEARECAAYFAjxNd+gACgkQFQV2RfX5MwRaZgCeK0t84vkxc9ZlHoHW5cKs26mD
ai0AoPXTBg6MKxd+o/CvXJ6+YyvntvH1
=Hert
-----END PGP SIGNATURE-----
```

Die Option „clearsign“ kopiert die Datei nach „name.asc“ und hängt die Signatur im Klartext an. Die mit „cat“ angezeigte Datei enthält den Inhalt und die Signatur.

```
conrad:~ # gpg --detach-sign eMail
```

Die Option „detach-sign“ ermöglicht es eine abgetrennte Signatur mit dem Namen „name.sig“ zu erzeugen. Dieses ist wichtig, wenn man zum Beispiel ein binäres Programm signieren möchte, das nicht verändert werden darf.

```
conrad:~ # gpg --verify eMail.asc
gpg: Signature made Wed Jan 03 05:32:08 2002 CET using DSA key ID F5F93304
gpg: Good signature from "Sven Tantau (testkey) <tantau@informatik.uni-bonn.de>"
conrad:~ #
```

„verify“ ist die Option zum prüfen der Signatur.
Die weiteren Optionen von „GnuPG“ zu Erkunden, bleibt dem Leser überlassen.

7.3 Probleme in der Praxis

In diesem Teil möchte ich die Probleme behandeln, die in der Praxis vorkommen und nicht nur von theoretischer Natur sind.

Eines der größten Probleme ist, dass Anwender ihre Passphrase vergessen. Wenn der öffentliche Schlüssel kein Verfallsdatum hat, gibt es keine Möglichkeit[16], den Schlüssel wieder für ungültig zu erklären.
Wenn der geheime Schlüssel zum Beispiel durch einen Festplattenfehler verloren geht, entstehen die gleichen Probleme. Daher ist es sehr Sinnvoll beim Generieren der Schlüssel ein Wiederufszertifikat zu erstellen und an einem sicheren Platz zu lagern (z.B. Bankschließfach).

Ein weiteres Problem ist die Benutzerfreundlichkeit von Verschlüsselungsprogrammen, die die meißten Anwender zurückschrecken lässt. Da der Mißbrauch des geheimen Schlüssels eine große Gefahr ist, ist eine gewisse Sorgfalt im Umgang mit den Programmen notwendig, die erst erlernt werden muss.

Die Gewissheit, dass ein öffentlicher Schlüssel zu der Person gehört, deren Name darin verzeichnet ist, ist oft nicht gegeben, da keine Trustcenter benutz wurden.

Es ist ein seltenes Problem, aber es kommt vor, dass ein Benutzer-Schlüssel mit einem falschen Namen verschickt wird und so leicht den Ruf des Namensträgers schädigen kann.

[16] ausser mit Wiederrufs-Zertifikat

Der Inkompatiblität verschiedener Software wird versucht durch offene Standards entgegenzutreten. Abgesehen von der Inkompatibilität der Kryptografie-Software, neigen einige eMail-Programme oder Text-Editoren dazu Daten zu verändern, was ein erfolgreiches Verifizieren unmöglich macht.

Unbemerktes Entwenden und Mißbrauchen des geheimen Schlüssels, bei einer Signatur-Methode ohne Fail-Stop-System, kann dazu führen, dass der Besitzer einen Beweis erbringen muß, der schwer Möglich ist. Öffentliche Computer zum Beispiel am Arbeitsplatz, sind oft unzureichend gegen widerrechtliche Benutzung gesichert.
Eine digitale Signatur unterscheidet sich von der herkömmlichen Signatur insofern, als dass es Möglich ist, dass A an den geheimen Schlüssel von B kommt.Jetzt ist es für A kein Problem verschiedene Daten als „A“ zu signieren, ohne dass B es merkt. Dieser Fall ist bei der normalen handschrifftlichen Unterschrifft ausgeschlossen.

8 Zusammenfassung

Auf der Abbildung 1 ist das Grundprinzip der Public-Key Signatur unter einbeziehung einer dritten Partei dargestellt.

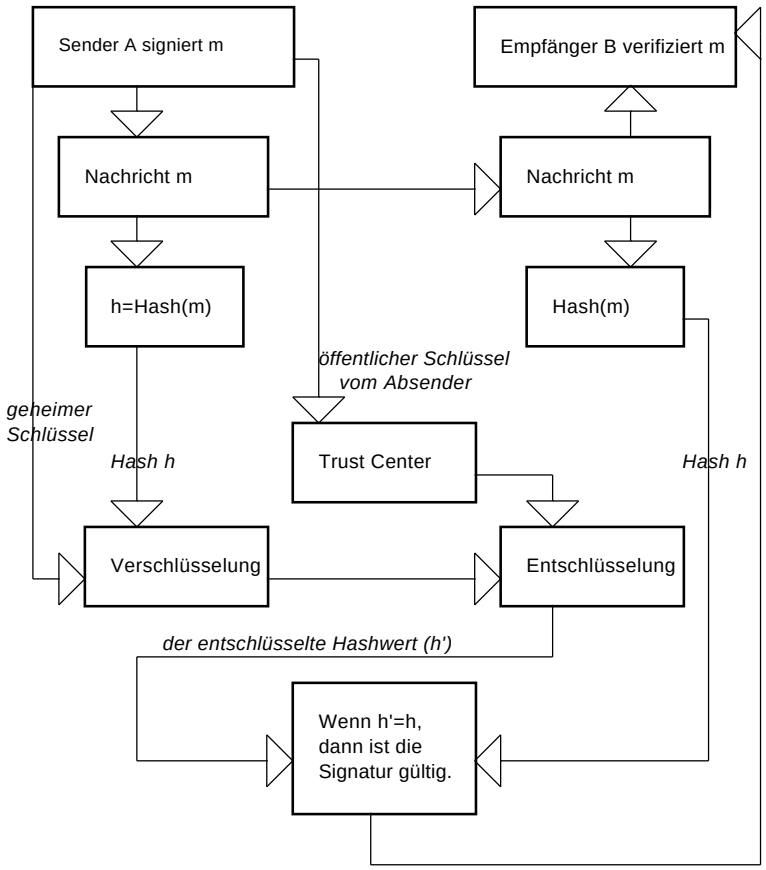

Abbildung 1: Digitale Signatur mit Public-Key-Verfahren und TTP

9 Ausblick

Die digitale Signatur wird in der vernetzten Gesellschafft immer wichtiger. Politisch gesehen ist die digitale Signatur eine Zwickmühle. Da Machthaber großen Wert darauf legt, die Kommunikation der Bürger uneingeschränkt, heimlich und automatisiert Überwachen zu können, wird in vielen Ländern diskutiert[17], ob man Kryptografie Regulieren sollte. Auch in Deutschland. Die Tatsache, dass Sichere Kryptografie für die digitale Signatur Notwendig ist, wird leicht vergessen.

Das Problem der Implementation der Software ist meiner Meinung nach das bedeutenste, da ein Sicherheits-System nur so sicher ist, wie der Teil des Systems, der am unsichersten ist. Sobald ein System zur digitalen Signatur von der Mehrheit der Computerbenutzer angewendet wird, werden sicherlich Programme entstehen, die auf übliche Implementationsprobleme angesetzt sind und andere Menschen schädigen.
Auch wenn die Kryptografie-Methoden und der geheime Schlüssel zum Beispiel auf einen Chip übertragen werden, gibt es niemals eine Sicherheit gegen Angriffe[18]. Es ist zu erwarten, dass sich in der Realität einem Angreifer immer Möglichkeiten bieten werden, genügend aber nicht unvorstellbare Energie vorrausgesetzt, eine Signatur zu fälschen. Der Glaube vieler Anwender, die Chance selber Opfer zu werden sei gering, ist in anbetracht des Automatismus mit dem Angreifer vorgehen, ein fataler Irrglaube.

Die digitale Signatur und ihrer Art der Implementierung nach dem heutigem Stand ist meiner Meinung nach wohl geeignet für private Kommunikation, aber ich betrachte zum Beispiel Bundesweiten Amtsgängen per Internet[19] mit Hife von Signatur-Methoden als gefährlichen Schritt in die verstärkte Abhängigkeit von Technik.

[17] Beispiel: In Frankreich ist Kryptografie, die der Staat nicht „lesen“ kann verboten

[18] Boneh, DeMillo und Lipton demonstrierten 1997 einen Angriff auf Chipkarten, bei dem durch elektrische Störung bestimmte Registerbits Invertiert werden konnten. Es ergab sich eine neue Möglichkeit der Krypoanalyse.

[19] ein Vorhaben das von der Bundesregierung gefördert wird

Literatur

[HAC] A. Menezes, P. van Orschot, S. Vanstone — Handbook of Applied Cryptography (http://www.cacr.math.uwaterloo.ca/hac)

[CFAQ] verschiedene Autoren — deutsche Übersetzeung der RSA-Crypto FAQ (http://www.iks-jena.de/mitarb/lutz/security/cryptfaq/)

[WST] William Stalling — Sicherheit im Datennetz

[CCC] Chaos Computer Club — verschiedenste Publikationen (https://www.ccc.de)